시스템 프로그래밍

Assignment2-3

Class : A

Professor : 김태석 교수님

Student ID : 2022202065

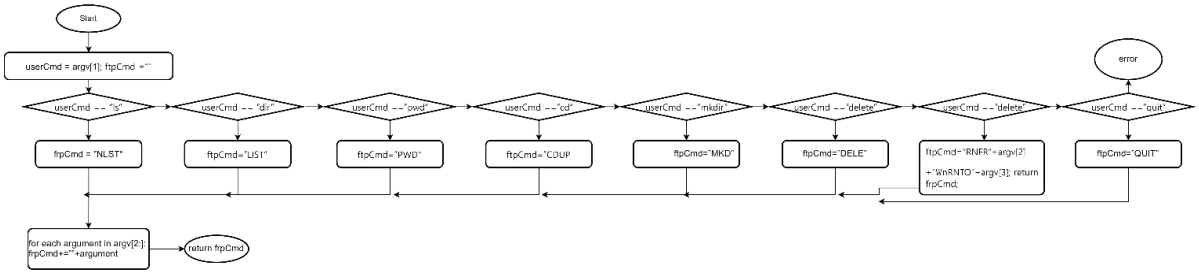
Name : 박나림

Introduction

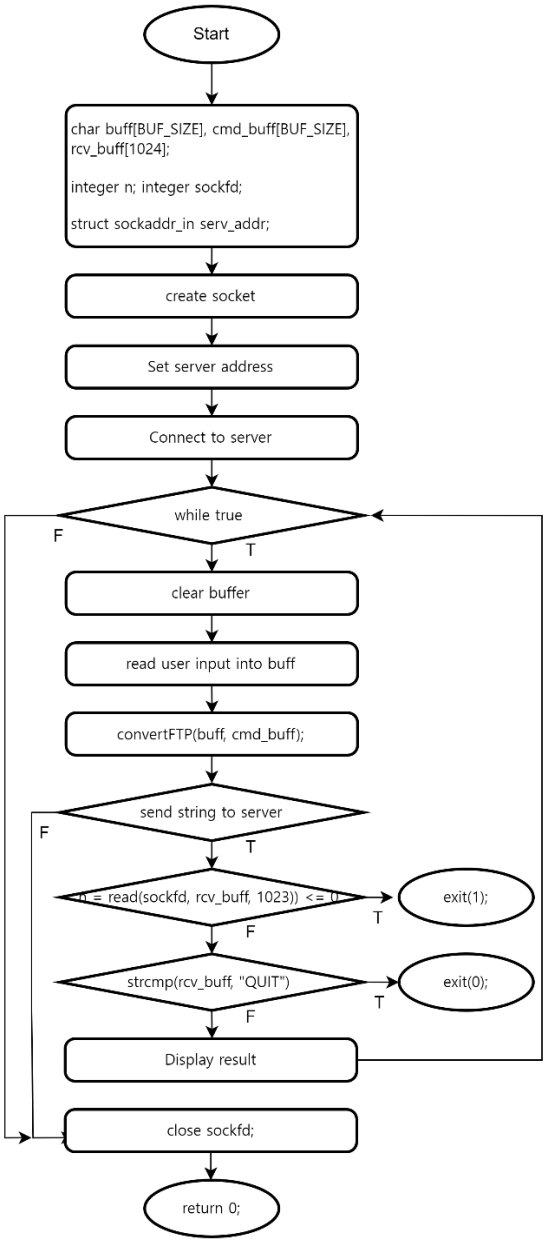
FTP 를 구현하는 프로젝트 중 fork, 다중 접속 시그널 처리를 다루는 과제이다. 기존에 구현했던 9 개의 user command(ls, pwd, dir, cd, mkdir, delete, rmdir, rename, quit)를 FTP command 로 변환하여 실행하는 프로그램에 소켓 알고리즘을 추가 구현하는 것이 목표이다. 서버는 클라이언트로부터 연결 요청이 들어올 때마다 fork 를 통해 child process 를 만들어서 명령어들을 수행한다. 이때 명령어 결과는 클라이언트로 보내져서 출력되며, 서버에서는 변환된 명령어 이름과 해당 명령을 수행중인 child process 의 PID 를 출력하도록 한다. 또한 이러한 클라이언트는 다중 접속이 가능하므로, 10 초마다 현재 실행중인 프로세스를 출력하도록 해야 한다. 마지막으로, 각 프로세스가 종료될 때 시그널 처리도 같이 하여 종료된 프로세스는 출력하지 않도록 구현한다.

Flow chart

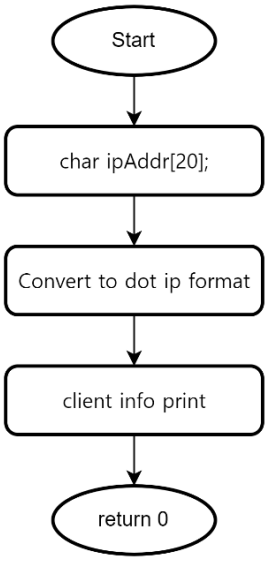
cli.c – convertFTP



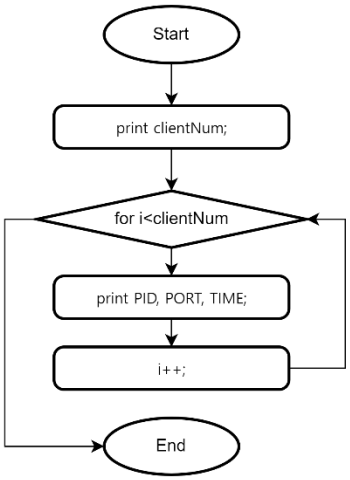
cli.c – main



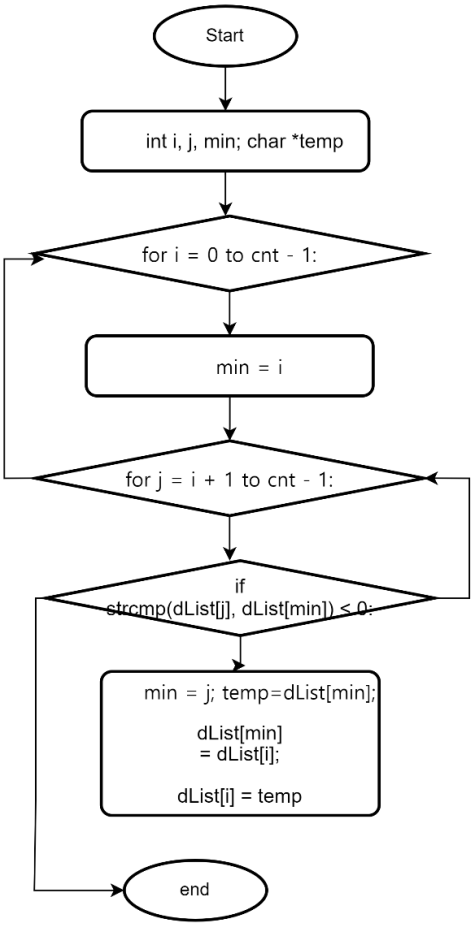
srv.c – client_info



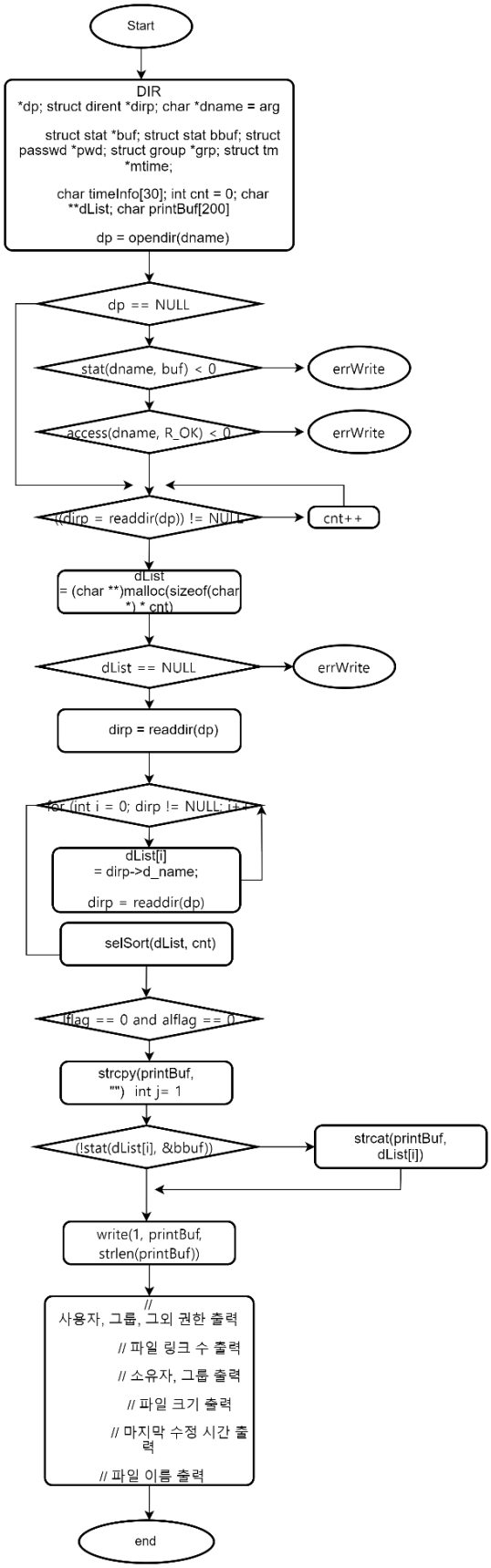
srv.c – currentCliInfo



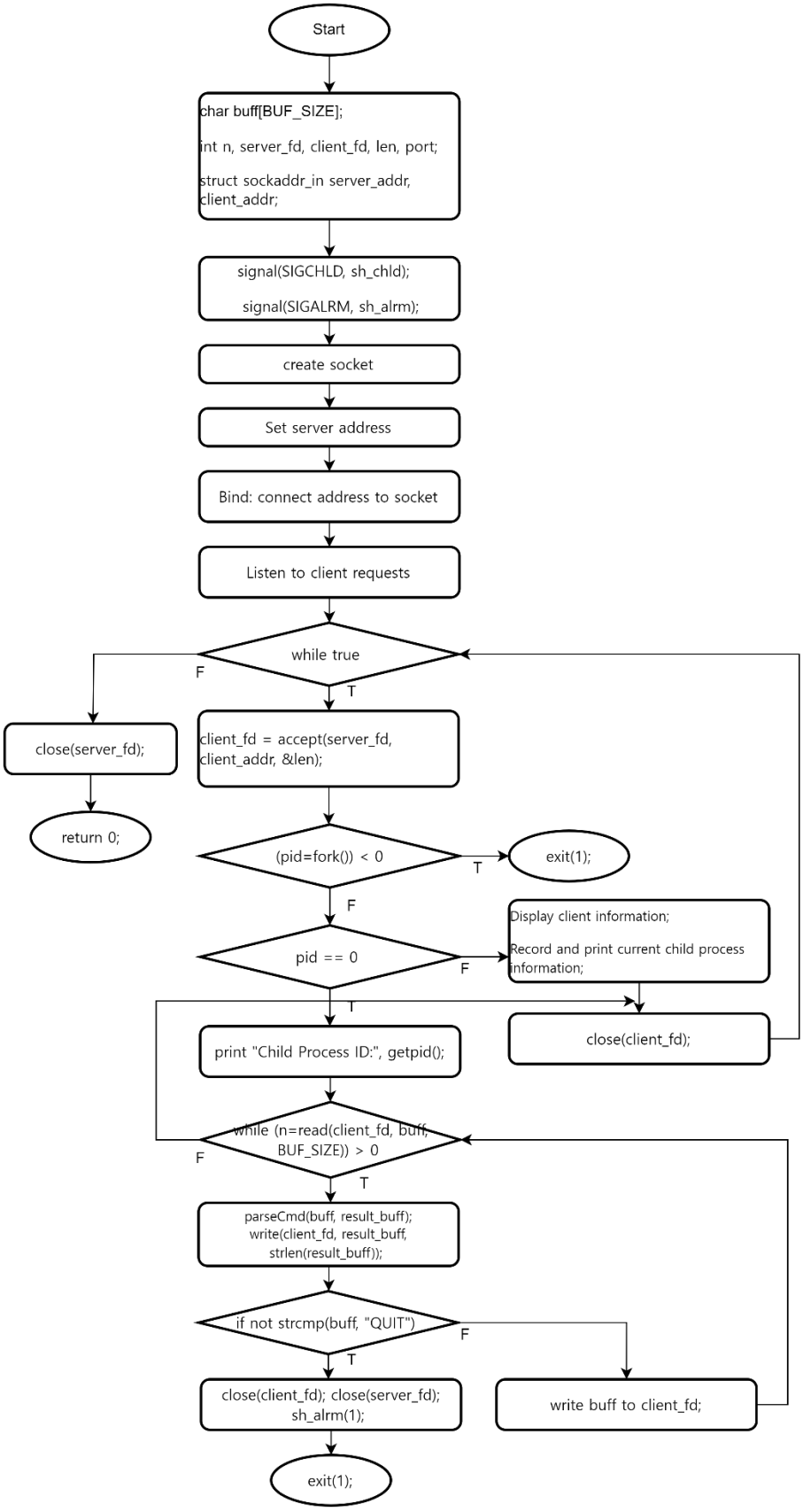
srv.c – selSort



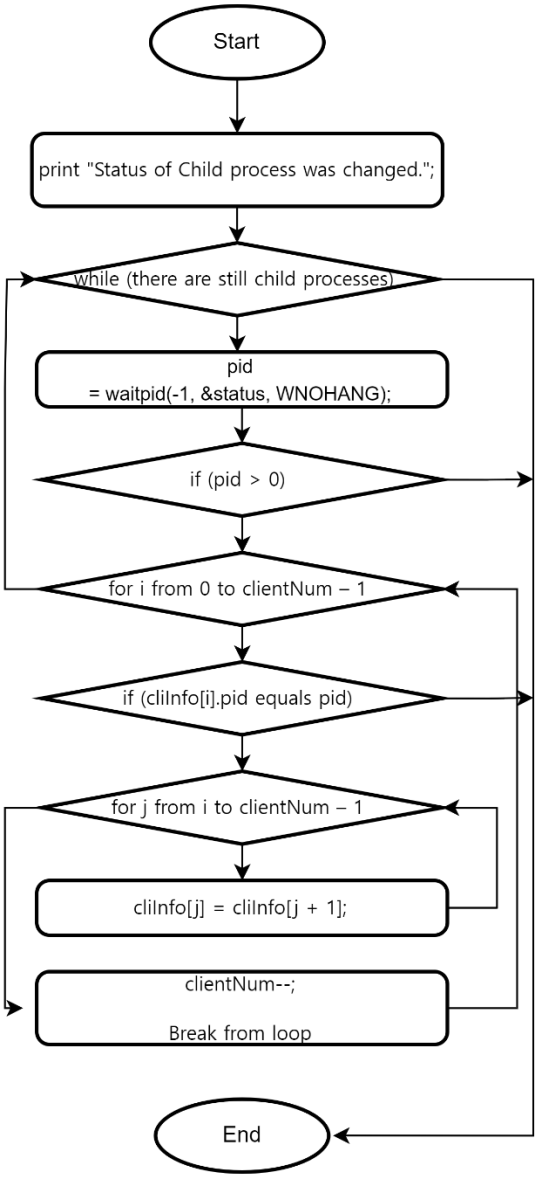
srv.c – lscmd


Start
DIR
*dp; struct dirent *dirp; char *dname = arg
struct stat *buf; struct stat bbuf; struct passwd *pwd; struct group *grp; struct tm *mtime;
char timeInfo[30]; int cnt = 0; char **dList; char printBuf[200]
dp = opendir(dname)
dp == NULL
stat(dname, buf) < 0
errWrite
access(dname, R_OK) < 0
errWrite
((dirp = readdir(dp)) != NULL)
cnt++
dList = (char **)malloc(sizeof(char *) * cnt)
dList == NULL
errWrite
dirp = readdir(dp)
for (int i = 0; dirp != NULL; i++)
dList[i] = dirp->d_name;
dirp = readdir(dp)
selSort(dList, cnt)
lflag == 0 and alflag == 0
strcpy(printBuf, "") int j= 1
(!stat(dList[i], &bbuf))
strcat(printBuf, dList[i])
write(1, printBuf, strlen(printBuf))
//
사용자, 그룹, 그외 권한 출력
// 파일 링크 수 출력
// 소유자, 그룹 출력
// 파일 크기 출력
// 마지막 수정 시간 출력
// 파일 이름 출력
end

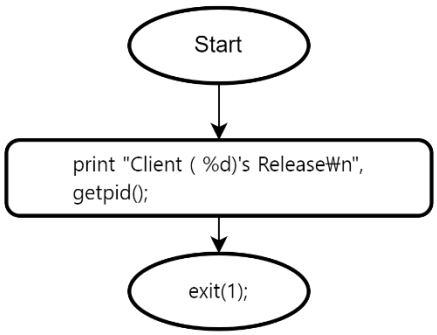
srv.c – main()


Start
char buff[BUF_SIZE];
int n, server_fd, client_fd, len, port;
struct sockaddr_in server_addr, client_addr;
signal(SIGCHLD, sh_chld);
signal(SIGALRM, sh_alrm);
create socket
Set server address
Bind: connect address to socket
Listen to client requests
while true
F
T
close(server_fd);
return 0;
client_fd = accept(server_fd, client_addr, &len);
(pid=fork()) < 0
T
exit(1);
F
pid == 0
F
Display client information;
Record and print current child process information;
T
print "Child Process ID:", getpid();
close(client_fd);
while (n=read(client_fd, buff, BUF_SIZE)) > 0
F
T
parseCmd(buff, result_buff);
write(client_fd, result_buff, strlen(result_buff));
if not strcmp(buff, "QUIT")
F
T
close(client_fd); close(server_fd); sh_alrm(1);
write buff to client_fd;
exit(1);

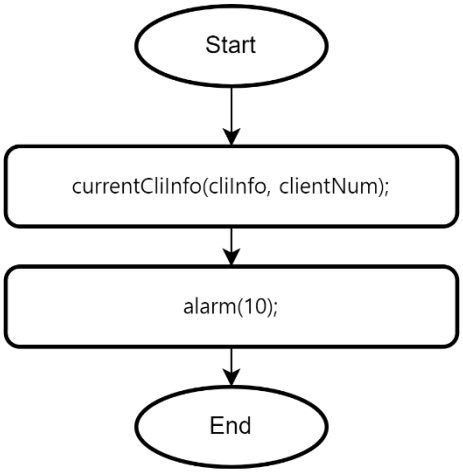
srv.c – sh_chld



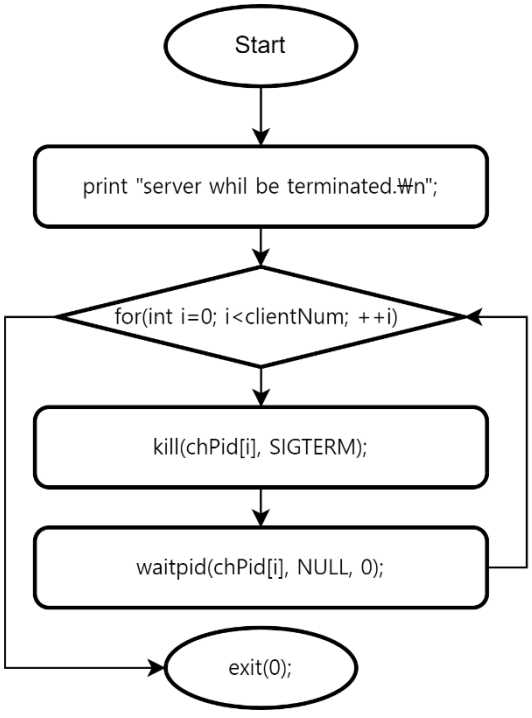
srv.c – sh_alrm



srv.c – sh_alrm_info



srv.c – sh_int



Pseudo code

cli.c

```
void errArg() {
        printf "Error: another argument is required₩n";
        exit;
}
```

```
void convertFTP(string buff, string cmd_buff) {
    char token = strtok(buff, " ");          // user command
    char userCmd = token;
    // Convert user command to FTP command
    if userCmd equals "ls" then
        copy "NLST" to cmd_buff;             // FTP command
        while (token = strtok(NULL, " ")) is not NULL          // option, argument parse
            add " " to cmd_buff;
            add token to cmd_buff;
    else if userCmd equals "dir" then
        copy "LIST " to cmd_buff;
        token = strtok(NULL, " ");           // path or name
        add token to cmd_buff;
    else if userCmd equals "pwd" then
        copy "PWD" to cmd_buff;
    else if userCmd equals "cd" then
        token = strtok(NULL, " ");
        if token equals ".." then
            copy "CDUP .." to cmd_buff;
        else
            copy "CWD " to cmd_buff;
            add token to cmd_buff;
        if token is NULL then
```

```
            errArg();                       // no path, error
    else if userCmd equals "mkdir" then
        integer errnum = 0;
        copy "MKD" to cmd_buff;
        while (token = strtok(NULL, " ")) is not NULL // directory name
            add " " to cmd_buff;
            add token to cmd_buff;
            increment errnum;
        if errnum equals 0 then
            errArg();                       // no directory name, error
    else if userCmd equals "delete" then
        integer errnum = 0;
        copy "DELE" to cmd_buff;
        while (token = strtok(NULL, " ")) is not NULL // directory name
            add " " to cmd_buff;
            add token to cmd_buff;
            increment errnum;
        if errnum equals 0 then
            errArg();                       // no file name, error
    else if userCmd equals "rmdir" then
        integer errnum = 0;
        copy "RMD" to cmd_buff;
        while (token = strtok(NULL, " ")) is not NULL   // directory name
            add " " to cmd_buff;
            add token to cmd_buff;
            increment errnum;
        if errnum equals 0 then
            errArg();                       // no directory name, error
    else if userCmd equals "rename" then
        token = strtok(NULL, " ");          // old name
```

```
        if token is NULL then
            errArg();                       // no change name, error
        else
            copy "RNFR " to cmd_buff;
            add token to cmd_buff;
            add "₩nRNTO " to cmd_buff;  // Add newline
            token = strtok(NULL, " ");      // new name
            add token to cmd_buff;
    else if userCmd equals "quit" then
        copy "QUIT" to cmd_buff;
    else // Command Error
        print "Error: invalid command";
        exit(1);

    integer n = length of cmd_buff;
    cmd_buff[n] = '₩0';             // set string end
}
```

```
int main(integer argc, array argv) {
    char buff[BUF_SIZE], cmd_buff[BUF_SIZE], rcv_buff[1024];
    integer n;
    integer sockfd;
    struct sockaddr_in serv_addr;
    sockfd = socket(AF_INET, SOCK_STREAM, 0); // create socket
    // Set server address
    set serv_addr to 0;
    serv_addr.sin_family = AF_INET; // set family
    serv_addr.sin_addr.s_addr = inet_addr(argv[1]); // set ip address
    serv_addr.sin_port = htons(atoi(argv[2])); // set port
    // Connect to server
    connect(sockfd, serv_addr, sizeof(serv_addr));
```

```
// Read user input and send to server, receive results and print
write "> " to STDOUT_FILENO;
while true
    clear buff;
    clear cmd_buff;
    read user input into buff;
    clear newline characters;
    if newline != NULL then
        set string termination;
    convertFTP(buff, cmd_buff); // user command -> FTP command
    n = length of cmd_buff;
    if write(sockfd, cmd_buff, n) > 0 then // send string to server
        if (n = read(sockfd, rcv_buff, 1023)) <= 0 then // receive and print string again
            write "Error: read() error!" to STDERR_FILENO;
            exit(1);
        end if
        rcv_buff[n] = '₩0'; // set string end
        if not strcmp(rcv_buff, "QUIT") then
            write "Program quit!!" to STDOUT_FILENO;
            exit(0);
        end if
        // Display result
        write rcv_buff to STDOUT_FILENO;
        write "₩n" to STDOUT_FILENO;
        write "> " to STDOUT_FILENO;
    else // quit if the string cannot be sent to the server
        break;
    end if
end while
close(sockfd);
```

```
    return 0;
    }
```

srv.c

```
int client_info(struct sockaddr_in *client_addr) {
        char ipAddr[20];        //ip address
        convert to dot ip format;
        printf Client info
        return 0;
}
```

```
void currentCliInfo(clientInfo cliInfo[], int clientNum) {
        printf clientNum;
        printf PID PORT TIME;
        for (int i=0; i<clientNum; i++) {
                printf PID PORT TIME;
        }
}
```

```
void selSort(array dList[], integer cnt) {
    integer i, j, min; char temp;
    for i = 0 to cnt - 1 { // Selection Sort
        min = I;
        for j = i + 1 to cnt - 1 {
            if strcmp(dList[j], dList[min]) < 0 then {
                min = j;
            }
        }
        temp = dList[min];
        dList[min] = dList[i];
        dList[i] = temp;
    }
```

```
void lsCmd(int aflag, int lflag, int alflag, char arg, char result_buff) {
    DIR *dp;
    struct dirent *dirp;
    string dname = arg;
    struct stat *buf ;// dList
    struct stat bbuf; // dList[i]
    struct passwd *pwd; // User ID
    struct group *grp; // Group ID
    struct tm *mtime; // Modified time
    char timeInfo[30]; // Time information
    int cnt = 0; // File count
    array dList; // Directory list
    char printBuf[200]; // Buffer for print
    Open directory;
    Counting directory list;
    rewinddir(dp); // Reset read position
    dList = malloc(sizeof(string) * cnt); // Dynamic allocation of directory list
    if dList == NULL then {
        errWrite("empty directory")
    }
    dirp = readdir(dp);
    Read again from the beginning and save to the list;
    selSort(dList, cnt); // Sorts the directory list by selection sort
    // Execute the ls command by option
    if lflag == 0 and alflag == 0 then // No-option or -a
        strcpy(printBuf, ""); // Print buffer initialization
        int j = 1; // Variables for print 4 or 5 per line
        for integer i = 0; i < cnt; i++ { // Print the entire list
            No-option, ignore hidden file continue;
            j++;
```

```
                if not stat(dList[i], &bbuf) then { // Distinguish between directory and file
                    strcat(printBuf, dList[i]);
                    if Directory, add to '/';
                    else strcat(printBuf, "₩t");
                }
            }
            strcpy(result_buff, printBuf); // Save ls result
        else // -l, -al
            for integer i = 0; i < cnt; i++ {
                -l, ignore hidden files continue;
                if not stat(dList[i], &bbuf) then
                    save directory property;
            }
        }
        set string end;
        close directory;
        free(dList);
    }
```

```
void parseCmd(input_buffer, result_buffer) {
    copy input_buffer to renameBuf;
    remove newline character from input_buffer;
    token = get next token from input_buffer;
    ftpCmd = token;
    initialize userCmd, printBuf, result_buff;
    initialize arg, aflag, lflag, alflag;
    if ftpCmd is "NLST":
        set userCmd to "ls";
        set arg to ".";
        process remaining tokens for options and arguments;
        call lsCmd function;
```

```
    if ftpCmd is "LIST":
        set userCmd to "dir";
        set arg to ".";
        process remaining tokens for options and arguments;
        call lsCmd function;
    if ftpCmd is "PWD":
        get current directory path and store it in result_buffer;
    if ftpCmd is "CWD" or "CUDP":
        change directory to the input path;
        get the current directory path and store it in result_buffer;
if ftpCmd is "MKD":
        process remaining tokens for directory names;
        create directories with the specified names;
    if ftpCmd is "DELE":
        process remaining tokens for file names;
        delete the specified files;
    if ftpCmd is "RMD":
        process remaining tokens for directory names;
        remove the specified directories;
    if ftpCmd is "RNFR":
        extract old and new file names from the input buffer;
        check if the new file name already exists;
        rename the file if it doesn't exist;
    if ftpCmd is "QUIT":
        set result_buffer to "QUIT";
 }
```

```
int main(int argc, char **argv) {
   Initialize variables, including buffers and socket addresses;
   Set signal handlers for SIGCHLD, SIGALRM, and SIGINT;
   Create a server socket;
```

```
Bind the server socket to the specified port;
Listen for incoming client connections;
While true {
    Accept incoming client connections;
    //Fork a child process to handle each client
    if(pid==0) { //child process
        close(server_fd);
        Print the child process ID;
        while(read client's command > 0) {
            parseCmd(buff, result_buff); //Execute after separating commands
            write(client_fd, result_buff, strlen(result_buff)); //send to client
            if(!strcmp(buff, "QUIT")) { //program quit
                    close(client_fd); close(server_fd); sh_alrm(1); //1 second }
        else { //parent process
          Display client information;
          Record and print current child process information;
          Set an alarm to print current child process information every 10 seconds;
        }
        close(client_fd);
}
close(server_fd);
return 0;
}
```

```
void sh_chld(signum) {
    print "Status of Child process was changed.";
    pid_t pid; int status;
    while (there are still child processes) {
        pid = waitpid(-1, &status, WNOHANG);
        if (pid > 0) {
          for i from 0 to clientNum – 1 {
```

```
                if (cliInfo[i].pid equals pid) {
                    for j from i to clientNum – 1 {
                        cliInfo[j] = cliInfo[j + 1];
                    }
                    clientNum--;
                    Break from loop
            }
}
```

```
void sh_alrm(int signum) {
            print "Client ( %d)'s Release₩n", getpid();
            exit(1);
}
```

```
void sh_alrm_info(int signum) {
            currentCliInfo(cliInfo, clientNum);
            alarm(10);
}
```

```
void sh_int(int signum) {
            print "server whil be terminated.₩n";
            for(int i=0; i<clientNum; ++i) {
                        kill(chPid[i], SIGTERM);
                        waitpid(chPid[i], NULL, 0);
            }
            exit(0);
}
```

결과화면

첫번째 클라이언트로 접속한 후, ls 명령어들과 pwd 를 실행한 결과이다.
클라이언트에서는 명령어를 수행한 결과가 출력된다.

```
kw2022202065@ubuntu:~/FTP_2-3$ ./cli 127.0.0.1 20000
> ls
Makefile        cli     cli.c   srv
srv.c

> ls -a
./      ../     Makefile        cli
cli.c   srv     srv.c

> ls -al
drwxrwxr-x      2       kw2022202065    kw2022202065    4096    05 11 20:40    ./
drwxr-xr-x      25      kw2022202065    kw2022202065    4096    05 11 20:40    ../
-rw-rw-r--      1       kw2022202065    kw2022202065    72      05 11 00:14    Makefile
-rwxrwxr-x      1       kw2022202065    kw2022202065    17464   05 11 20:40    cli
-rw-rw-r--      1       kw2022202065    kw2022202065    6009    05 11 20:40    cli.c
-rwxrwxr-x      1       kw2022202065    kw2022202065    31592   05 11 20:38    srv
-rw-rw-r--      1       kw2022202065    kw2022202065    18695   05 11 20:38    srv.c

> pwd
"/home/kw2022202065/FTP_2-3" is current directory
```

서버에서는 먼저 연결된 클라이언트의 정보를 출력한다. 이때 새로 연결된 직후에는 ip, port 정보와 함께 현재 클라이언트의 개수, 실행중인 자식 프로세스의 정보가 함께 출력된다. 그리고 수행되는 FTP 명령어 이름과 함께 해당 명령어를 실행하는 자식 프로세스의 PID 값이 출력된다. 또한 실행중인 프로세스의 정보들은 10 초마다 다시 출력된다.

```
kw2022202065@ubuntu:~/FTP_2-3$ ./srv 20000
==========Client info==========
client IP: 127.0.0.1

client port: 32996
==============================
Current Number of Client: 1
 PID    PORT    TIME
5459    20000   0
Child Process ID: 5459
> NLST  [5459]
Current Number of Client: 1
 PID    PORT    TIME
5459    20000   10
> NLST -a       [5459]
> NLST -al      [5459]
Current Number of Client: 1
 PID    PORT    TIME
5459    20000   20
> PWD   [5459]
Current Number of Client: 1
 PID    PORT    TIME
5459    20000   30
Current Number of Client: 1
 PID    PORT    TIME
5459    20000   40
```

두번째 클라이언트로 접속한 후, quit 을 실행한 결과이다. 클라이언트에서는 종료 메시지와 함께 종료된다.

```
kw2022202065@ubuntu:~/FTP_2-3$ ./cli 127.0.0.1 20000
> quit
Program quit!!
```

서버에서는 새로 연결된 클라이언트의 ip, port 정보와 실행중인 자식 프로세스가 출력된다. 이때 첫번째 프로세스도 실행중이므로 함께 출력된다. 이후 QUIT 을 전달받으면 종료 메시지와 함께 해당 클라이언트의 정보가 삭제된다. 따라서 10 초 뒤 출력되는 현재 프로세스 정보에서 두번째 클라이언트가 삭제된 것을 볼 수 있다.

```
5459    20000   40
==========Client info==========
client IP: 127.0.0.1

client port: 53206
================================
Current Number of Client: 2
 PID    PORT    TIME
5459    20000   40
5469    20000   0
Child Process ID: 5469
> QUIT  [5469]
Client ( 5469)'s Release
Status of Child process was changed.
Current Number of Client: 1
 PID    PORT    TIME
5459    20000   50
```

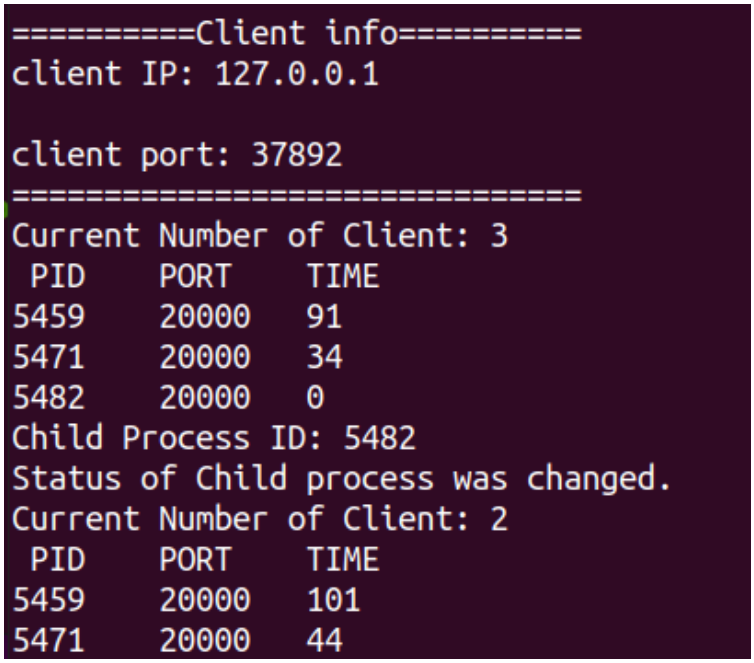
세번째 클라이언트로 접속한 후, mkdir 과 ls 를 실행한 결과이다.

```
kw2022202065@ubuntu:~/FTP_2-3$ ./cli 127.0.0.1 20000
> mkdir test


> ls
Makefile        cli     cli.c   srv
srv.c   test/
```

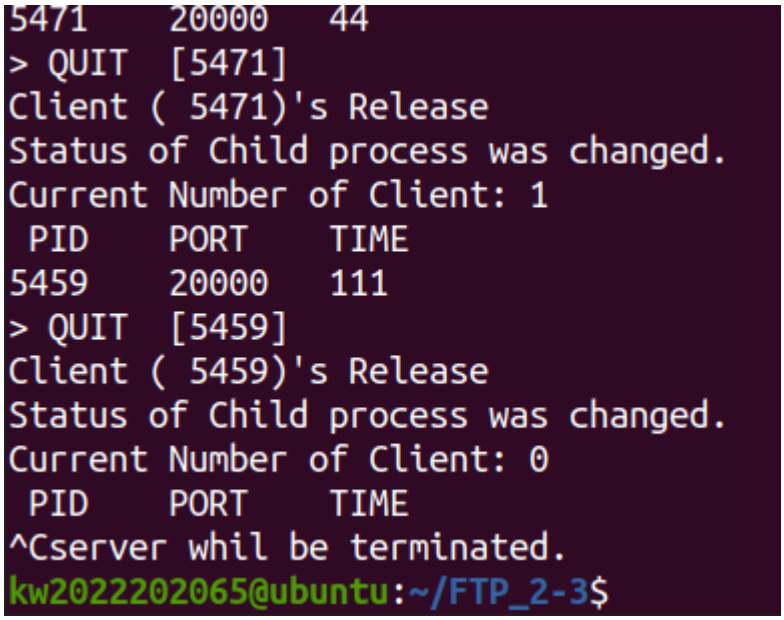
서버에서는 세번째 클라이언트 정보와 함께 명령어 내용, 실행중인 프로세스(첫번째, 세번째) 정보가 10 초마다 출력된다.

```
5459    20000   30
==========Client info==========
client IP: 127.0.0.1

client port: 40168
===============================
Current Number of Client: 2
 PID    PORT    TIME
5459    20000   57
5471    20000   0
Child Process ID: 5471
Current Number of Client: 2
 PID    PORT    TIME
5459    20000   67
5471    20000   10
> MKD test      [5471]
Current Number of Client: 2
 PID    PORT    TIME
5459    20000   77
5471    20000   20
> NLST  [5471]
Current Number of Client: 2
 PID    PORT    TIME
5459    20000   87
5471    20000   30
```

네번째 클라이언트로 접속하면 마찬가지로 실행중인 프로세스 정보들이 출력된다. 네번째 프로세스는 ctrl+c 로 종료하였다. 그러면 10 초 뒤에 네번째 프로세스가 삭제된 상태로 출력된다.



이후 두번째, 첫번째 프로세스들을 차례대로 quit 한 결과이다. 프로세스 정보들이 삭제되는 것을 볼 수 있다. 마지막에는 프로세스들이 더 이상 나타나지 않는다. 이후 서버에서 ctrl+c 로 종료하면, SIGINT handler 가 동작하여 메시지와 함께 모든 클라이언트, 자식 프로세스를 종료시킨다.



고찰

이번 과제는 기존에 구현해놨던 Assignment#1 에 소켓 프로그래밍을 추가하는 것이었는데, 기존에 했었던 클라이언트 프로그램에서 명령어를 분리하는 함수를 수정해야 했었다. 원래는 argv 를 이용하여 분리하였었는데, 이번에 while 문으로 반복하면서 받으려면 기존 방법을 사용할 수 없었기 때문이었다. 그래서 strtok()을 이용해서 분리시키도록 수정하였다.

명령어 부분에 관해서 코드를 완성시킨 뒤 실행해보니, 클라이언트에서 명령어 결과가 출력되지 않았다. 이에 코드를 다시 확인해보니, read 하는 과정에서 괄호가 잘못 들어가 있었다. 수정하니 정상적으로 결과를 읽을 수 있었는데, 이번엔 다른 명령어들이 잘 실행이 되지 않았다. 서버로 FTP 명령어를 전달하는 것부터가 잘못 되었었다. 특히 NLST 를 한 뒤 PWD 를 하면 PWDT 처럼 추가 문자가 들어가서 명령어가 수행이 되지 않았다. 이는 버퍼 관리가 문제라 생각하여, 클라이언트와 서버에서 문자열 끝 설정을 다 해주었으나 똑같이 수행이 되지 않았었다. 다른 명령어도 실행해보니, 명령어를 실행할 때마다 버퍼에 그 전 명령어 결과가 남아 있는 것을 확인하였다. 그래서 버퍼를 사용하는 코드 중간마다 memset()을 활용하여 버퍼를 비워주는 코드를 추가하였더니, 결과가 정상적으로 출력될 수 있었다. 저번 과제까지는 문자열 끝 설정하는 것을 중요시 여겼는데, 이번 과제를 통해 버퍼를 비워줘야 한다는 것도 깨달았다.

다중 저속 허용 & 시그널 처리 부분에서도 여러 시행착오가 있었다. 현재 실행 중인 클라이언트의 정보를 출력하는 것을 처음에는 자식 프로세스에서 하도록 코드를 작성하였다. 그 결과, 프로세스가 추가될 때마다 그전 정보들이 제대로 저장이 되지 않았다. 정보를 담는 구조체를 전역변수로 선언했음에도 결과가 제대로 나오지 않았다. 알고 보니 자식 프로세스에서 진행할 시 그 정보는 다른 자식 프로세스에서 접근을 못하였다. 따라서 부모 프로세스에서 진행하도록 코드를 옮긴 후 실행하니 정보들이 남게 되었는데, 문제는 pid 가 제대로 나오지 않았다. 이에 대해서도 다시 공부해보니 자식 프로세스가 아니라 부모 프로세스에서 pid 를 사용하려면 getpid()가 아니라 그전에 저장했던 pid 를 써야했었다. 이렇게 수정하니 정보가 잘 출력되었다. 하지만 프로세스를 종료할 때 해당 정보가 삭제되지 않았었는데, 원래는 quit 에서 종료할시 해당 구조체의 인덱스를 줄이는 것으로 코드를 짰었다. 마찬가지로 다시 공부를 해 보니 그렇게 하면 제대로 삭제할 수 없음을 깨달았다. 그래서 sh_chld 함수를 통해 해당 시그널을 받으면 waitpid 를 통해 종료된 프로세스를 검색하여 구조체에서 삭제하는 방식으로 수정하였다.

이번 과제를 통해 버퍼 관리, 다중 접속 정보 다루기, 시그널 처리 등 소켓 프로그래밍에 대해 더 자세히 공부해볼 수 있었다.

Reference

강의자료 참고